Advanced Analysis of Algorithms - Quiz I (Solutions)

L. Kovalchick
LCSEE,
West Virginia University,
Morgantown, WV
{lynn@csee.wvu.edu}

1 Problems

1. We discussed the *Hat Paradox* problem in class. To recap, N men throw their hats in a ring and after a random shuffling, each man draws out a hat. We showed that the expected number of people who get their own hats is 1, irrespective of the number of people involved. Derive an upper bound on the probability that all the men involved get their own hats?

   Solution:
   Let the random variable X represent the number of people that get their own hat. Observe that X ranges from 0 to N. We want an upper bound on the probability that everyone gets his own hat (i.e., $Pr\{X = N\}$). Using Markov's Inequality, we have $Pr\{X \geq N\} \leq \frac{E[X]}{N}$. Observe that this system can be rewritten as $(Pr\{X = N\} + Pr\{X > N\}) \leq \frac{E[X]}{N}$. Notice that $Pr\{X > N\} = 0$, since it is not possible for more than N men to get their own hats. Thus, we have the following.

$$(Pr\{X = N\} + Pr\{X > N\}) \leq \frac{E[X]}{N}$$
$$Pr\{X = N\} \leq \frac{1}{N}, \qquad \text{since } E[X] = 1$$

   □

2. Consider the Towers of Hanoi problem. There are 3 pegs, viz., P_1, P_2 and P_3. On P_1 there is collection of N disks, such that the circumference of each disk is larger than the disk above it. The goal is to move the N disks from P_1 onto P_2, subject to the following constraints:

   (a) In one move, you can move precisely one disk from one peg to another peg.

   (b) At no time, can a disk be placed on another disk with a smaller circumference.

   Argue with an example, that the additional peg, P_3, is *necessary*. Derive a recurrence relation for calculating the number of moves required to move N disks from P_1 onto P_2. Solve the recurrence, i.e., obtain a closed-form expression for the recurrence derived.

   Solution:
   Notice that when we have only one disk, since we do not need P_3, we can move the disk directly from P_1 to P_2.
   Now, consider the case when we have 2 disks. If we are only allowed to use 2 pegs (i.e., P_1 and P_2) we would first move the disk on the top of P_1 to P_2. Observe that this is the only choice that we could have made because according to constraint (a), in a single move we can only move a single disk from one peg to another. After the first move, according to the definition of the problem, we have the disk with the smaller circumference on P_2 and the disk with the larger circumference on P_1. Now, we are stuck, because we

cannot move the disk on P_1 onto P_2, on account of constraint (b). Thus, the only move that we can make is to move the disk on P_2 back onto P_1, which gives us the same problem that we started with. Therefore, P_3 is necessary, if the number of disks is greater than 1.
Next, we must show that P_3 is sufficient (i.e., we only need a total of 3 pegs to solve the problem for N disks, $N \geq 1$). Let $T(N)$ be the minimum number of moves needed to solve the problem for N disks. Notice that $T(1) = 1$ (i.e., if we have 1 disk, then only a single move is needed). The interesting case occurs when $N \geq 2$. Notice that when the number of disks is N such that $N > 1$, we must first move $N - 1$ disks from P_1 to P_3 using P_2 as an auxiliary peg. Then, we can move the Nth disk from P_1 to P_2 using only a single move (i.e., $T(1) = 1$). We then move the remaining $N - 1$ disks from P_3 to P_2 using P_1 as an auxiliary peg. Therefore, the existence of one more peg (i.e., P_3) is sufficient to solve the problem.
Observe that we have the following recurrence.

$$\begin{aligned} T(1) &= 1 \\ T(N) &= 2 \cdot T(N-1) + 1, \qquad \text{for } N > 1 \end{aligned}$$

We will now guess that $T(N) = 2^N - 1$. We will prove that our guess is correct using induction on N.
Base Case:
Our base case occurs when $N = 1$. Substituting $N = 1$ into $T(N) = 2^N - 1$ we have the following.

$$\begin{aligned} T(1) &= 2^1 - 1 \\ &= 2 - 1 \\ &= 1 \end{aligned}$$

Thus, $2^N - 1 = T(1)$ and the base case is true.
Inductive Hypothesis:
Assume that $T(N-1)$ is true, i.e., $T(N-1) = 2^{N-1} - 1$.
Inductive Step:
We must prove that $T(N)$ is true. Using our recurrence, we have the following.

$$\begin{aligned} T(N) &= 2 \cdot T(N-1) + 1 \\ &= 2 \cdot (2^{N-1} - 1) + 1 \qquad \text{using our inductive hypothesis} \\ &= 2^N - 2 + 1 \\ &= 2^N - 1 \end{aligned}$$

Thus $T(N-1) \rightarrow T(N)$; applying the principle of Mathematical Induction, we conclude that the closed form expression for our recurrence is $2^N - 1$ for $N \geq 1$. □

3. Let **A** denote an array of n elements. Describe a strategy that finds the maximum and minimum of the array, using at most $\frac{3}{2}n$ comparisons.

Solution:
Consider the array **A** of n elements. Assume first that n is even. We first break the array into $\frac{n}{2}$ pairs of 2 elements each. For each pair, we compare the elements to each other and place the smaller of the two into the set S_{min} and the larger of the two into the set S_{max}. Observe that if the maximum element of **A** is in S_{min}, then it is also in S_{max} since corresponding to each element in S_{min}, there is at least one element in S_{max} that is greater than or equal to it. Accordingly, we can focus on S_{max} for finding the maximum element of **A**. Using an identical argument, we can focus on S_{min} for finding the minimum element of **A**. The process of building S_{max} and S_{min} requires exactly $\frac{n}{2}$ comparisons.

Finding the maximum element of S_{max} requires exactly $\frac{n}{2} - 1$ comparisons; likewise finding the minimum element of S_{min} requires exactly $\frac{n}{2} - 1$ comparisons; thus the total number of comparisons to find both the

maximum and minimum of $\mathbf{A}$, when the number of elements is even, is at most $\frac{n}{2} + (\frac{n}{2} - 1) + (\frac{n}{2} - 1)$, which is at most $\frac{3}{2} \cdot n - 2$.

Now consider the case that n is odd. It follows that $(n-1)$ is even. Apply the same strategy discussed above to obtain the maximum (x_{max}) and minimum (x_{min}) elements of the first $(n-1)$ elements, using at most $\frac{3}{2}(n-1) - 2$ comparisons. Then using at most 3 more comparisons, determine the maximum and minimum elements of the set $\{x_{max},\ x_{min}\ A[n]\}$ to obtain the maximum and minimum elements of the complete array $\mathbf{A}$. The total number of comparisons needed is $\frac{3}{2}(n-1) - 2 + 3$ which is at most $\frac{3}{2} \cdot n$. □

4. Show that for any integer $n \geq 0$,

$$\sum_{k=0}^{n} \binom{n}{k} \cdot k = n \cdot 2^{n-1}.$$

Hint: What is $\sum_{k=0}^{n} \binom{n}{k}$*?*

Solution:
Let $S_n = \sum_{k=0}^{n} \binom{n}{k} \cdot k$, then we have the following.

$$\begin{aligned}
\sum_{k=0}^{n} \binom{n}{k} \cdot k &= \sum_{k=1}^{n} \binom{n}{k} \cdot k, \qquad \text{since for } k = 0 \text{ we get } 0 \\
&= \sum_{k=1}^{n} \frac{n! \cdot k}{k! \cdot (n-k)!} \\
&= \sum_{k=1}^{n} \frac{n \cdot (n-1)! \cdot k}{k \cdot (k-1)! \cdot (n-k)!} \\
&= n \cdot \sum_{k=1}^{n} \frac{(n-1)!}{(k-1)! \cdot (n-k)!} \\
&= n \cdot \sum_{k=1}^{n} \frac{(n-1)!}{(k-1)! \cdot (n-1-(k-1))!} \\
&= n \cdot \sum_{j=0}^{n-1} \frac{(n-1)!}{j! \cdot ((n-1)-j)!}, \qquad \text{since } j = k-1 \\
&= n \cdot \sum_{j=0}^{n-1} \binom{n-1}{j} \\
&= n \cdot 2^{n-1}, \qquad \text{since } \sum_{k=0}^{n} \binom{n}{k} = 2^n
\end{aligned}$$

Thus, $\sum_{k=0}^{n} \binom{n}{k} \cdot k = n \cdot 2^{n-1}$. □

5. Professor Smith proposes a new sorting algorithm called *Smith-Sort*

Function SMITH-SORT($\mathbf{A}, i, j$)
1: **if** $(A[i] > A[j])$ **then**
2: Exchange these elements in the array.
3: **end if**
4: **if** $((i+1) \geq j)$ **then**
5: **return**
6: **end if**
7: $k \leftarrow \lfloor \frac{j-i+1}{3} \rfloor$
8: SMITH-SORT($\mathbf{A}, i, j-k$)
9: SMITH-SORT($\mathbf{A}, i+k, j$)
10: SMITH-SORT($\mathbf{A}, i, j-k$)

Algorithm 1.1: SMITH-SORT

Prove that Algorithm (1.1) correctly sorts an array of n elements provided in $\mathbf{A}[i]$ through $\mathbf{A}[j]$. You may assume that all elements are distinct.

Proof: We use mathematical induction on the number of elements in the array, i.e., $n = |j - i + 1|$ to prove its correctness.

We have two base cases, $n = 1$ and when $n = 2$. Consider the case when $n = 1$. The array $\mathbf{A}$ contains only one element, thus $i = j$. The test $(A[i] > A[j])$ fails, since $A[i]$ is equal to $A[j]$. The test $(i + 1 \geq j)$ succeeds, since $i = j$. Thus, the algorithm correctly returns a sorted array when there is exactly one element.

Consider the case when $n = 2$. The test $(A[i] > A[j])$ will swap the elements of $\mathbf{A}$ if necessary and then the test $(i + 1 \geq j)$ succeeds, since $i = j - 1$. Thus, the algorithm correctly sorts a 2 element array.

Accordingly, we have proved that Algorithm (1.1) works correctly for the base cases.

Inductive Hypothesis: Assume that Algorithm (1.1) correctly sorts the elements $A[i]$ through $A[j]$ of the array $\mathbf{A}$, when $3 \leq |j - i + 1| \leq n$.

Consider the case when the array $\mathbf{A}$ contains $n + 1$ elements. Thus, $i = 1$ and $j = n + 1$, when SMITH-SORT() is called. The algorithm first calculates $k = \lfloor \frac{j-i+1}{3} \rfloor = \lfloor \frac{n+1-1+1}{3} \rfloor = \lfloor \frac{n+1}{3} \rfloor$. Notice that $i \leq k \leq j$, when $|j - i + 1| \geq 3$.

We divide the array $\mathbf{A}[i \cdot \cdot j]$ (conceptually) into the following regions: $L_1 : \mathbf{A}[i \cdot \cdot i + k]$, $L_2 : \mathbf{A}[i + k + 1 \cdot \cdot j - k]$ and $L_3 : \mathbf{A}[j - k + 1 \cdot \cdot j]$. Thus, the first recursive invocation, viz., SMITH-SORT($\mathbf{A}$,i,$j - k$) is called on $L_1 \cup L_2$, the second recursive invocation is called on $L_2 \cup L_3$, and the third recursive invocation is called on $L_1 \cup L_2$ again.

By the inductive hypothesis, the first recursive invocation will return a correctly sorted array, since the number of elements in $\mathbf{A}[i \cdots j - k]$ is at most n, when $n \geq 3$. It follows that after the first recursive invocation, each element in L_2 is at least as large as every element in L_1. Arguing similarly, we observe that the second recursive invocation correctly sorts the set $L_2 \cup L_3$; further every element in L_3 is at least as large as each element in L_2. From the correctness of the first recursive invocation, we can therefore conclude that each element in L_3 is at least as large as every element in $L_1 \cup L_2$; further as per the inductive hypothesis, all the elements in L_3 have been correctly sorted. The third recursive invocation completes the sorting procedure, since as per the inductive hypotheis, it correctly sorts $L_1 \cup L_2$ and thus the set $L_1 \cup L_2 \cup L_3$ is correctly sorted.

□